Volume 10 Issue 04 Published, November 9, 2006 ISSN 1535-864X DOI: 10.1535/itj.1004.05

Intel® Technology Journal

Autonomic Computing

Machine Learning for Adaptive Power Management

More information, including current and past issues of Intel Technology Journal, can be found at:
http://developer.intel.com/technology/itj/index.htm

Machine Learning for Adaptive Power Management

Georgios Theocharous, Corporate Technology Group, Intel Corporation
Shie Mannor, Department of Electrical and Computer Engineering, McGill University
Nilesh Shah, Corporate Technology Group, Intel Corporation
Prashant Gandhi, Corporate Technology Group, Intel Corporation
Branislav Kveton, Department of Computer Science, University of Pittsburgh
Sajid Siddiqi, School of Computer Science, Carnegie Mellon University
Chih-Han Yu, Department of Computer Science, Harvard University

Index words: power management, machine learning

ABSTRACT

Adaptive Power Management (APM) systems for laptops decide when to place a component into various power-saving states given the user activity. The long-term goal of an APM system is to maximize the battery life while minimizing the annoyance to the user. The state-of-the-art in commercial solutions is timeout policies. These policies switch components into their low power states based on thresholds of periods of inactivity. Unfortunately, such methods not only waste power during the periods of inactivity, but also needlessly annoy the user when they turn off components at inappropriate times. Research in APM, on the other hand, has focused more on modeling system dynamics and not on usage patterns. We propose a system that learns when to turn off components based on different user patterns. We describe the challenges of building such a system and progressively explore a range of solutions. We experiment with a direct approach that predicts when to turn off a component given usage features (e.g., historical keyboard and mouse activity, active application, history of network traffic, and history of CPU utilization). To improve performance of the direct approach we partition data based on the context and train the learning algorithms separately for each context. Context could be past idleness or any partitioning of the data that improves performance. We then propose a model-based approach that captures the temporal dynamic of system and user state, the cost of power and user annoyance, as well as the effect of power-saving actions on the user and system. Our direct approach is validated on a large data corpus collected from multiple real users where results show a considerable improvement over traditional timeout methods.

INTRODUCTION

Adaptive Power Management (APM) attempts to increase the perceived battery life of a laptop by turning off certain components when their services are not going to be needed. Of course, it is not known in advance which services will be needed and when. Moreover, the perceived impact of having certain services not available is much greater than the perceived impact of others. The goal of APM is to extend the battery life as much as possible with an acceptable perceived loss of performance. APM, as opposed to non-adaptive power management, attempts to make better decisions with time and to adapt to individual users.

APM is an example of an autonomic system. The autonomics concept is derived from the human body's autonomic nervous system that regulates individual organ function, and for the most part is not subject to voluntary control. An autonomic computing system controls the functioning of computer applications or systems without explicit user input. The goal of autonomic computing is to create systems that are self-managing, self-healing, and self-protecting. Autonomic computing promises to reduce expenditures associated with operations, maintenance, and support, and to significantly improve the end user's experience.

Autonomic systems ensure smooth and uninterrupted operation by monitoring the system state, diagnosing problems, identifying user context, and executing actions to fix the problems. For example, an APM system monitors the user and laptop state, diagnoses the user mood or context, and changes power states such that the user is not noticeably impacted. Autonomic systems need to naturally reason about uncertainty in both perception

and action. In an APM system, perception uncertainty arises from the fact that the user context cannot be directly observed from sensors, such as keyboard and mouse activity, or the currently active application. Actions that turn off components also generate uncertainty. They create uncertainty in terms of the time it takes to turn a component on and off and in terms of the effect it would have on the user's context. For example, placing the machine on standby does not always take the same amount of time and sometimes the Operating System (OS) does not even allow it. If the user context is for example a measure of idleness, then going to standby, could either increase idleness if it succeeds or decrease it if it fails, hence the uncertainty. If the user context is a measure of the users' mode of operation (e.g., working, browsing for fun) then going to standby could again change the context in an unpredictable manner.

Modeling real-world phenomena as stochastic processes and sequential decision-making under uncertainty (see Figure 1) are widely used paradigms in artificial intelligence. The process of estimating these stochastic process models and the computation of optimal policies fall into the category of machine learning. Machine learning refers to the problem of constructing computer programs that automatically improve with experience. The simplest class of machine learning algorithms are *classifiers,* which are functions that learn (from examples) to map input patterns to output classes. For the APM domain, our algorithms learn to map laptop usage patterns (at the OS level) into power management actions as shown in Figure 2.

In this paper, we first summarize past approaches to the APM problem and then describe a machine-learning approach that maps user patterns to power-saving sections. Our approach outperforms current commercial solutions such as timeout policies. We then propose a more general model-based approach that relies on building stochastic process models. We finally discuss the potential application of such models to other domains.

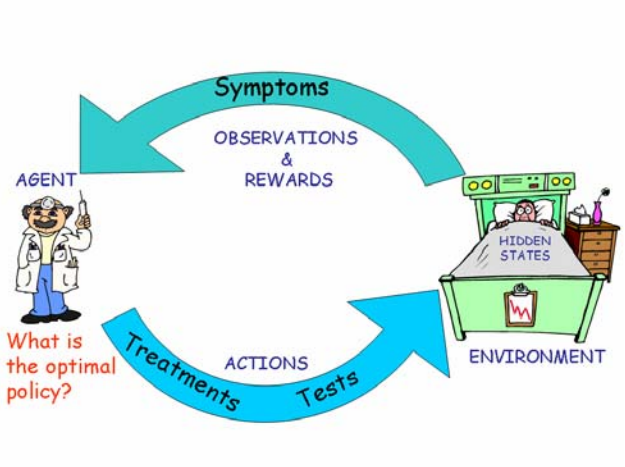

Figure 1: A classic sequential decision-making task under uncertainty. The doctor performs treatments and diagnostic tests on the patient, which change in a stochastic manner the internal state of the patient. The doctor then observes symptoms. The optimal policy is one where in the long run the patient is made well.

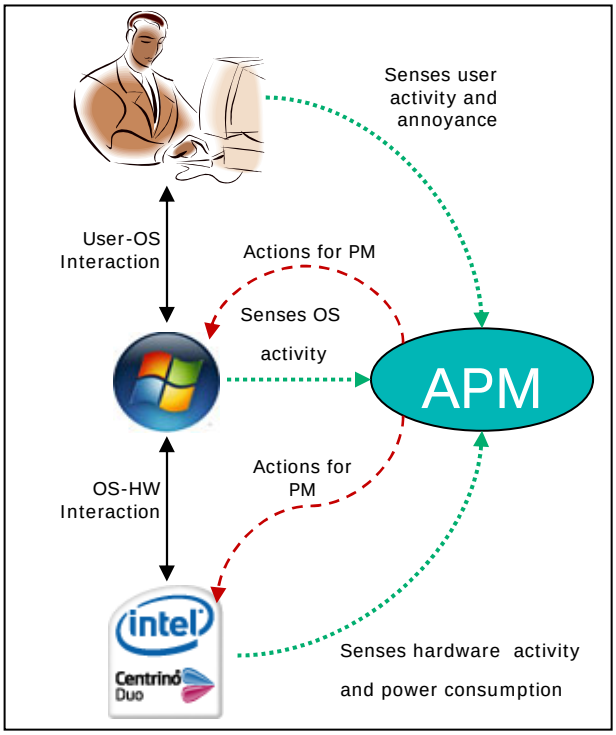

Figure 2: APM takes as input user and system activity and issues power management actions to the OS and to the hardware

ADAPTIVE POWER MANAGEMENT

An APM system maps the computer system (OS and hardware) activity and the user activity into power management actions as shown in Figure 2. The goal of such a system is to maximize battery life while minimizing the impact on the user. In this section, we describe the state-of-the-art in APM solutions and our proposition for explicit consideration of the user's context and perception.

Past Approaches

Currently used power management techniques are based on *timeout policies*. These policies shut down a component after it is not used for some predefined time. The aggressiveness of the scheme is reflected by the length of the timeout before a specific component is shut down. For example, Windows* OS has several built-in power management schemes that allow the user to choose between different levels of aggressiveness. The user is expected to switch manually between these schemes. Timeout policies are fairly simple and robust. They are, however, non-adaptive and may be too fast or too slow to react.

Applying machine-learning techniques to the APM problem is in its infancy. In [16] a simple predictive approach for deciding if to turn a component on or off was investigated. The authors attempt to predict the length of the idle period and claim that typically a short idle time is followed by a long active time and vice-versa (i.e., the prediction is based on the recent history). The decision rule eventually recommended that the component be shut down if it was used for less than some threshold value. They explored two approaches: a threshold value that is determined using regression and a threshold value that is manually obtained from data. In both approaches, the threshold parameters were obtained based on data in a non-adaptive manner. The approach of [7] is to predict the future delay as an exponentially weighted sum of recent delays. Our work uses learning techniques that focus more on the adaptive and dynamic aspects of power management.

Several papers considered policy optimization in the context of power management ([1], [14], [15]). According to this approach, a single Markov Decision Process (MDP) or a Semi-Markov Decision Process (SMDP) is constructed and solved using linear programming. An MDP is a stochastic process model of the state of the world and costs of actions (see Figure 6). An SMDP is an MDP where the next state does not only depend on the current state but also on how long the current state has been active. The Markov property, which states that the current state is sufficient for predicting the next state, is violated, hence the semi-Markov terminology. The Markov models used in these works are extremely simple and their state space is mostly an active/not active indication. State transitions are estimated from data and the optimization is done off-line. A somewhat more complex stochastic optimization model is presented in [12]. The model used in this work is a non-stationary process where there are several modes. For each mode, the process is a Markov decision process. In general, these models are not realistic because they assume that the state variables can be directly observed from sensors. Our work differs from this line of work in that we explicitly take into consideration the user activity (usually not directly observable) and also consider the perceived performance.

User-Based APM

The papers mentioned above attempt to use dynamic models estimated from data. To the best of our knowledge, they are all based on relatively small and simulated amounts of data. Most importantly, these works do not model user state in their decision making, which we believe to be essential.

Monitoring user activity (i.e., context) from sensor observations is a well-studied problem, e.g., see [6]. However, there are relatively few instances where activity monitoring is linked to decision making, which is the goal of APM. In this paper, we focus on making the actual decision: which component to turn off and when. With this goal in mind, we introduce a novel representation of context, namely the idleness duration variable, to describe the state of the user. Though simple, this is an effective way to characterize context for this decision problem in the power management domain. We also address the issue of how the user perceives the performance degradation (a measure we call *annoyance*).

Reducing the power consumption to the bare minimum can be easily done by moving to standby mode on every occasion. This, however, is not very useful since a power management system applying such a policy will not be very usable. It is clear that a "good" policy should minimize power consumption. Therefore, one has to consider the tradeoff between the power savings and the perceived performance degradation, i.e., the annoyance. In order to examine this tradeoff, we look at tradeoff-type plots, where one axis shows the power saving and the other shows the level of annoyance. With no power savings, all the components are on all the time and the annoyance is assumed to equal zero. As the annoyance increases, the power savings may increase. Still, the power savings will never be more than the minimal power consumption level needed to perform the required operations. Figure 3 demonstrates a power-saving annoyance curve. Different users may choose different points on the power-saving annoyance curve according to their desired tradeoff.

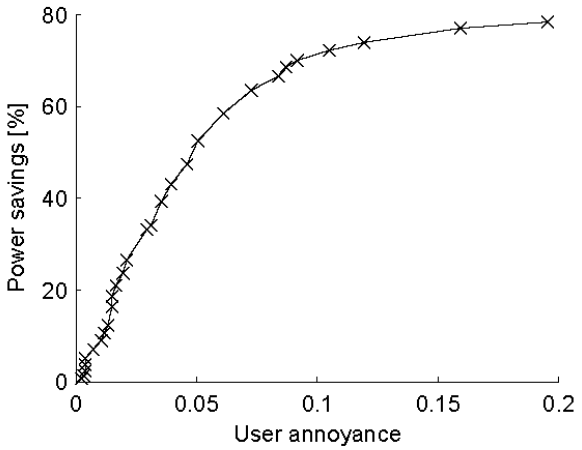

Figure 3: A tradeoff curve for power savings versus annoyance. The power savings cannot pass a certain threshold as shown with the converging curve to be approximately 80%. This curve was produced from a real trace, where the points on the curve represent various timeout policies. The furthest top right point represents a timeout parameter of 1 second and the furthest lower left, a timeout parameter of 600 seconds.

Quantifying the power savings is straightforward and it can be done in Watts per second, or the proportion of time a component is off. In our experiments, we considered four components: turning off the LCD, turning off the WLAN, running the CPU in low frequency, and placing the laptop in standby mode. When the laptop is turned on or in standby mode, it consumes 18.5 and 0.7 Watts per second, respectively. Turning off the LCD, or the WLAN, or switching the CPU into its low-frequency mode, results in the consumption of 14, 17.5, and 15 Watts per second, respectively.

Quantifying the annoyance is trickier. Essentially, the problem is that annoyance is a subjective measure and different users may feel a different level of annoyance for the same sequence of actions. Based on interviews with users, we decided to measure the annoyance as follows. If the system decided by mistake to turn off the CPU, WLAN, and LCD, it leads to annoyance of 1, 3, and 7, respectively. Moving to a standby mode by mistake leads to annoyance of 10. We know if turning off a component was a mistake since we can detect if this component is needed or not after it was turned off. For example, if we turn the LCD off and after a few seconds the user opens a new application and clicks the mouse we can safely assume that turning the LCD off was a mistake. The different annoyance coefficients represent a collective and subjective estimate of how much more annoying is turning off one component compared to turning off another.

THE DIRECT APPROACH

In our direct policy approach, we trained a separate classifier for each action. The input is the sensor measurements and features, and the output is whether to turn the component on or off.

The sensors that were sampled every second were active application, keyboard and mouse activity, CPU load, and network traffic. Out of these sensors we constructed various features, such as sums, decayed sums, averages, and gradients of sensor readings over different past windows.

The labels for the classification problems were generated by heuristic laws after going through the trace of sensor readings. We used several ad-hoc rules to determine the labels, such as, if turning WLAN off and packets are sent within 30 seconds, determines the action to turn WLAN off to be a mistake. This labeling scheme gives a "target" policy that is guaranteed not to annoy the user because it will never turn off components when needed in the next fixed look-ahead.

As a yardstick for a learning algorithm, we used logistic regression, k-nearest-neighbors and the C4.5 decision tree. These algorithms are simple to use and produced more or less similar results. Since these algorithms are standard, we do not describe them here. We refer the reader to [5]. Many classifiers (such as logistic regression) have some threshold function. For example, if the outcome is probabilistic between the two classes, then the threshold is the minimum probability for deciding the first class. By varying the threshold, we can make the resulting classifier more or less aggressive leading to different points on the annoyance-power-saving curve.

The second yardstick we used is timeout-based policies. The decision to turn off a component in such a policy is according to the time passed since it was last used–we turn it off if it was not used for more than some predefined threshold. We can control the level of annoyance by varying the threshold (the larger the threshold the less aggressive the timeout policy and the less annoyance we will observe).

Context-Based Policy Learning

For context-based policy learning, we explicitly take into consideration the user context variable. The idea is to separate the data for each value of the context variable and then train a separate classifier for each data partition. We then choose the decision threshold for each classifier such that the overall power savings are maximized. As we show in our experiments, this approach performs better than naïve classifiers trained on all data.

In general, user context is not directly observable and represents the user's internal state, mood, or intentions. In our experiments we discovered that the time since the component was last active is a good feature for predicting the actions. Therefore, we defined context to be the time since the component was last active. We partitioned these durations into 30 categories, where each category represents a value for the context variable. Value 1 means in the previous time step the component was active, and value 30 means the component was idle for the last 600 steps. We chose the rest of the 28 thresholds heuristically across the 600 seconds. We observed that the longer a component is idle the more likely it will be idle (which means it should be easier to predict the action); therefore, for larger idle times we made the partition density coarser than that for shorter idle times. Empirically, the number 30 gave us reasonable performance. For smaller numbers the predictions were not as good, and for larger numbers we were over fitting.

In order to determine the thresholds for each context classifier we used an optimization algorithm. The algorithm starts by setting the least annoying thresholds for all classifiers. At each step, we increase the threshold that corresponds to the maximal power savings over annoyance increase ratio subject to a global annoyance constraint. Identifying the threshold and its new value is computationally simple because we only enumerate a single parameter and freeze the remaining ones. This optimization method proved efficient and it always converged to a local optimum. Figure 4 shows a graphical representation of our algorithm.

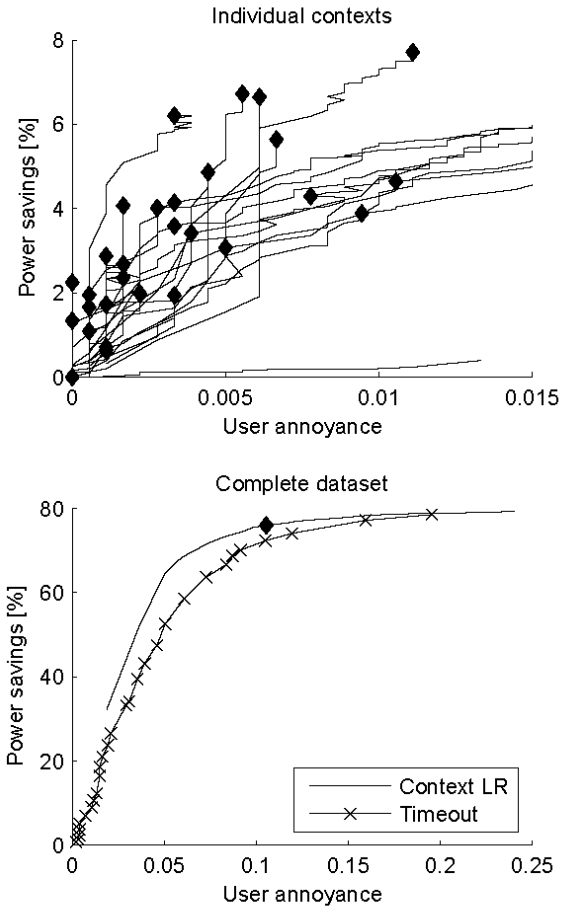

Figure 4: The top graph shows power savings/user annoyance curves corresponding to the 30 individual contexts. The black diamonds represent the decision points returned by the heuristic optimization method subject to the annoyance constraint of 0.1. The bottom graph compares power savings/user annoyance curves for the optimized mixture classifier (black line) and timeout policies (black line with cross markers). The black diamond denotes the global annoyance constraint for 0.1.

Experiments

We compare the quality APM policies with naïve classifiers and with timeout policies. The different policies were evaluated on 42 traces, which were collected for 7 users, and they represent the cumulative experience of 210 usage hours. The data are obtained from T42 laptops by using an application that we developed. We present results for standby policies only. These results generalize to the power management of the LCD, CPU, and WLAN. In order to make a fair comparison, we compare the

performance of the various policies for the same annoyance level.

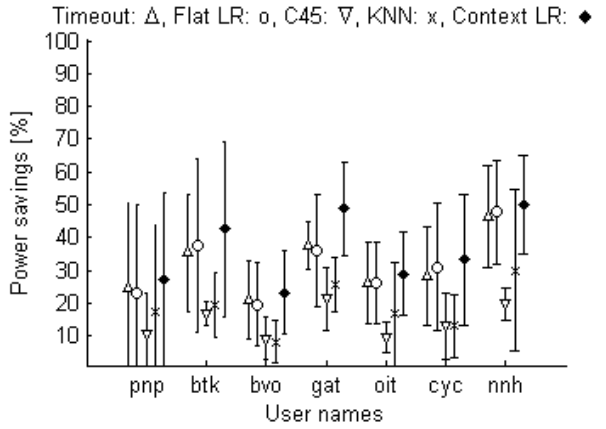

Figure 5: The graph shows expected power savings and their standard deviations for various classification techniques. The target annoyance level is 0.05.

Based on Figure 5, we know that the context-based LR outperforms the baselines methods. This observation justifies the benefits of training classifiers for individual contexts, which are constructed based on our domain knowledge. An obvious question that arises is why we save more power for some users than for others. This phenomenon can be explained from the distribution of the idle times for each user. Simply, if the user spends more time in longer inactivity periods, we save more power. This demonstrates the difference between users. Different users may have a very different behavior and the APM policy has to adapt to the specific user.

DISCUSSION

So far we have experimented and described a direct approach to the problem of APM. We believe that significant performance improvements might be possible by using stochastic process models of the domain. These models capture the temporal dynamics of user and system state as well as the annoyance and power costs. Having learning and planning algorithms could potentially be beneficial to other domains as we describe at the end of this section.

Model-Based Approach

The model-based approach decouples the decision-making process from the problem of learning and estimating the model of the environment. Learning a model refers to the process of defining/discovering domain variables and how they relate to each other. Using the model refers to the process of computing decisions given the model. Next we describe three models of increasing expressiveness as well as increasing difficulty in learning them and using them.

Markov Decision Processes

The MDP model facilitates reasoning in domains where actions change the states stochastically and where there is usually a delayed reward signal (e.g., winning in backgammon is attributed to possibly many moves along the play and not only to the final move). Figure 6 gives a graphical MDP representation.

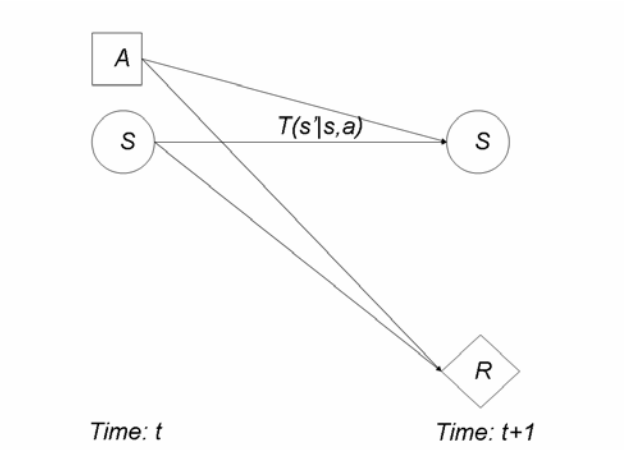

Figure 6: This is a graphical representation of an MDP model. The left column represents what happens at time t and the right column what happens at t+1. Arrows represent dependence.

In an MDP, there exists an underlying state that encompasses all the information needed to represent the world. This state is a summary of the relevant systems and models. Given the state of the system the future is independent of the past. This means that the true state captures all the information needed to describe the system. This last property is often referred to as the *Markov property*. The agent can act in response to the observed state. The result of the action depends on the state of the world and may be uncertain.

The objective of the agent is to maximize his long-term cumulative reward (typically, the reward of interest is the discounted future reward where the near future is more important than the more remote future). An immediate reward is received when exercising an action. This immediate reward depends on the action made by the agent and the true state. The immediate reward that is received in a particular state following a certain action may also be random.

Formally, MDPs are described as a 4-tuple: $<S,A,T,R>$ as shown in Figure 6. S denotes a finite set of states and A denotes a finite set of actions. $T(s'|s,a)$ denotes the transition probability from state s to state s' under action a. R(s,a) is the reward for taking action a in state s. In Figure 6 we see the dependencies between the different

elements that define the MDP: the current state (S) and action (A) determine the next state (S') according to the transition probability (T); the current state (S) and action (A) determine the reward (R).

The main advantage of the MDP model is its simplicity. On the one hand, learning the parameters and computing an optimal policy can be done efficiently by several different algorithms (e.g., [2]). On the other hand, the simplicity of the MDP model comes with a price: it cannot capture unobservable dynamics, and its success hinges on the assumption that the state of the system can be estimated with no errors.

Dynamic Bayesian Networks

A Dynamic Bayesian Network (DBN) describes a stochastic process where some of the variables are observed and some are not. Figure 7 shows a graphical representation of a special case of DBNs where there is a single variable called a Hidden Markov Model (HMM). In a DBN the agent reasons about the state of the process indirectly though the observed variables. For example, the doctor in Figure 1 makes an inference about the internal state of the patients through symptoms.

An HMM is described as a 4-tuple: $<S,T,Z,O>$ as shown in Figure 7. S denotes a finite set of states, and Z denotes a finite set of observations. T(s'|s) denotes the transition probability from state s to state s'. O(z|s) is the probability of observing the observation z in state s. In the figure we see the dependencies between the different elements that define the HMM: the current state (S) determines the next state (S') according to the transition probability (T); the observation (Z) depends on the current state (S) through the observation probability (O).

The advantage of DBNs is their ability to capture complex dynamics that are not completely observable. The disadvantage of DBNs is their lack of support for decision making. Estimating the DBN parameters can be done relatively efficiently using algorithms such as the Expectation-Maximization algorithm.

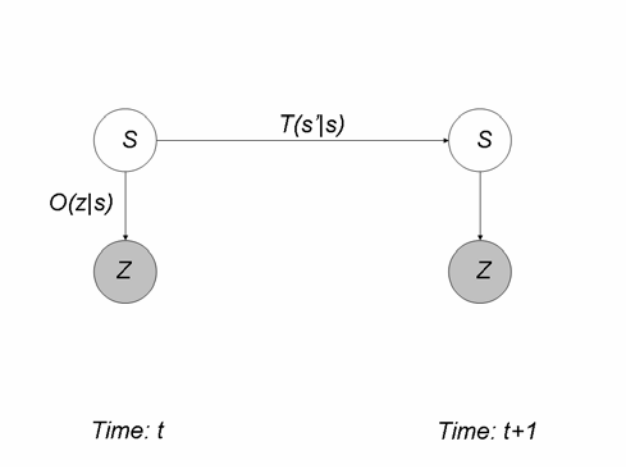

Figure 7: This is a graphical representation of a simple dynamic Bayesian networks called a Hidden Markov Model

POMDPs

The Partially Observable Markov Decision Process (POMDP) model is the combination of HMMs/ DBNs and MDPs. The idea behind the POMDP model is to combine the strengths of HMMs (capturing dynamics that depend on unobserved states) and MDPs (taking the decision aspect into account) without significantly sacrificing computational efficiency. The model is depicted in Figure 8, where it can be seen that the POMDP contains elements from both MDPs and HMMs. POMDPs are a natural conceptual framework for modeling sequential decision tasks as illustrated in Figure 1. A POMDP model describes sequential decision tasks under uncertainty. A control policy in a POMDP computes an action after every observation such that in the long-run (discounted or average) the expected utility is maximized. Uncertainty, therefore, has two sources in POMDPs. First, the true state of the world is usually unknown (the doctor does not know what the patient has exactly). Second, even if the state is known, some actions may have uncertain consequences (different treatments may work differently for patients with the same condition).

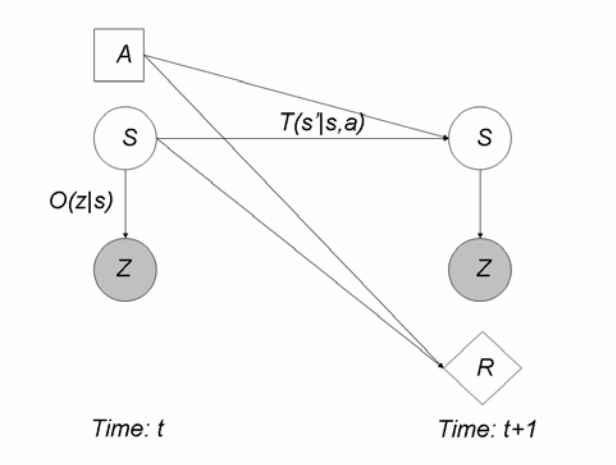

Figure 8: This is a graphical representation of a POMDP model. It is created by combining the MDP and HMM models.

A POMDP policy computes actions at every time step. Due to the fact that the system state is not observed, the POMDP policy maps actions to all possible probability distributions over the states, otherwise called *belief states B*. The belief state *b(s)* represents the agent's current belief that the true state is *s*. The goal of the policy is to maximize the long-term reward. The POMDP policy can be computed by solving the Bellman Equation [2]. Figure 9 illustrates the POMDP execution system. For the sake of simplicity we do not discuss how belief update is done or how the Bellman Equation is solved. A good introduction to POMDPs can be found in [8].

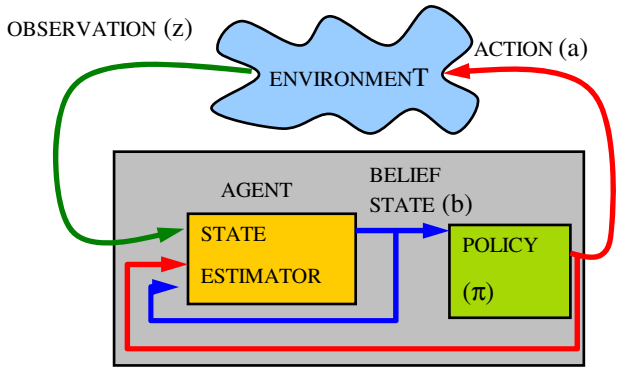

Figure 9: This figure represents the POMDP execution system. The state estimator module takes the previous belief states, the action taken and observation received, and produces a new belief state. The Policy module takes as input the belief states and outputs an action.

Some of the strengths and weaknesses when using the POMDP model are these:

- *Computing the policy*. In general, computing the optimal policy is intractable [11],[10]. Still, reasonable approximations can be made by following some simple heuristics. There are many algorithms for finding an adequate policy that may not necessarily be optimal.
- *Controlling the level of abstraction*. The POMDP model allows choosing different levels of abstraction leading to models that have more or less states depending on the POMDP designer preferences. This leads to a tradeoff between the amount of information that has to be supplied on the model (a large number of states would require a lot of information on the model) and the potential accuracy of the model (a large number of states would enable a higher accuracy).
- *Reasoning in terms of information*. Reasoning in terms of information allows the designer to distinguish between actions that maximize utility and actions that provide information on the state.
- *Handling model uncertainty*. Lack of knowledge regarding the specifics of a model can be naturally incorporated into the POMDP model. Similarly to DBNs, the model parameters can often be learned from data, using algorithms such as the Expectation-Maximization algorithm. This allows for the building of a rough initial guess to start with and refining it later, based on the observed data.

The Model-Based Approach to Adaptive Power Management

Among the different models presented above, the POMDP model is the only one that is rich enough to capture the two main aspects of APM. First, the world model includes a human user and a complex computer system that cannot be assumed to be perfectly observable. Second, APM management involves making decisions on which components to turn on and off.

There are many possible ways to define a POMDP for APM. The suggested POMDP model is shown in Figure 10. Casting this model in the language of POMDP we have this: the actions (A) are to turn on/off (for each component); the state space (S) is a combination of system state and user state; the possible observations are the various sensors/features we described before; the transition probability (T) and the observation probability (O) are both unknown a-priori and must be learned from data.

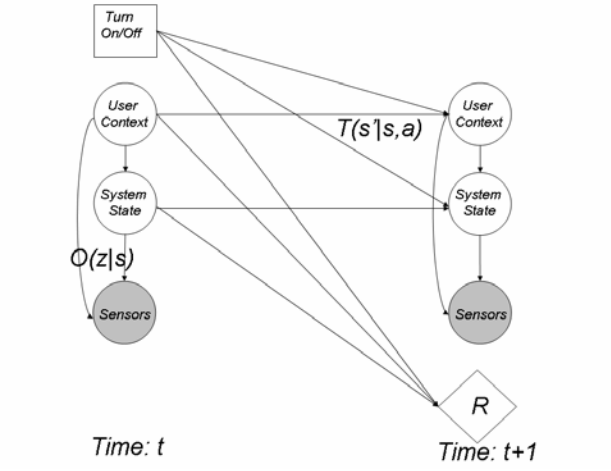

Figure 10: Our model has user, system, and sensor variables. The user variables are not directly observable and represent user context or activity. The system state indicates whether the component is idle or active. The sensor variables are keyboard and mouse activity, application running, CPU load, and network traffic. The reward function R represents power cost as well as cost incurred by annoying the user.

The main problem in constructing the POMDP for APM is how to construct the state space. This is a thorny problem at the core of applying POMDP methodology to any real-world problem. It is obvious that a model that fully describes the system or the user context (i.e., the way the user is interacting with the system) would be too complex to describe, and would certainly not lead to any useful computations. We therefore have to balance the complexity of such a model with our need to be able to obtain a useful APM policy.

We emphasize that learning a good POMDP model in general involves answering questions such as "what is the user's context," "how does that context relate to system state," "how does context change over time," and "what is the level of user annoyance." These questions are not trivial in the field of artificial intelligence. Still, we believe we can obtain partial answers to these questions based, in part, on heuristics or even arbitrary decisions. Once we have such answers we can offer a solution to the POMDP.

Our ongoing research focuses on finding an adequate model to represent a realistic system and then solving that model. We are currently studying several aspects of the model. First, we are developing a more complex model for user context (other than past idleness). In particular, we look at automatic context construction (that is generated by automatically clustering sequences). Second, we are looking at different ways to measure annoyance. Instead of using an annoyance measure that is the same for all users, we consider learning the annoyance from the user, based on individual feedback, and even prompt the user for explicit feedback. Third, we are studying the statistics of the duration between changes of the values of the system and user variables. Fourth, we consider the initial period where the system is initializing. The problem here is that when the system starts running the amount of information that is available is small and the decisions are potentially sub-optimal. In this initial stage there might be a lot of value in exploring actions that can provide a lot of information on the user. The problem of balancing between information gathering and using the information to perform optimally is known as the exploration-exploitation dilemma. As was shown in [13], model-based approaches may be unsatisfactory in their initialization phase.

Other Domains

There are many real-world problems that can be framed as sequential decision-making tasks under uncertainty. Stochastic models such as POMDPs offer a good framework for such tasks because they encompass the idea that there are underlying hidden states (user and system) and the idea that the system needs to constantly take actions under uncertainty to satisfy some long-term goals and to maximize some performance criterion.

Explicit reasoning about the user state is imperative in many systems that have significant interaction with the user. For example, in the power management domain, if a user does not mind interruptions we can afford to have more aggressive and less accurate power management policies, where accuracy is in terms of predicting whether a component can be turned off. The user state could be summarized by variables that are related to user activity (physical and mental) and utility preferences. Learning mental user models is not an inconceivable idea and has previously been explored in opponent modeling in games such as in [3] and [4].

The idea of user adaptive autonomic systems such as the APM systems could be extended to domains that go beyond platforms. Instead of an isolated computer platform we could call it an "environment" which has both user states and task states. The environment produces observations and utilities. The autonomic agents' task is to maximize their long-term utility as explained in Figure 1. Figure 11 and Figure 12 show some examples of similar domains previously studied by researchers. In each figure, we describe the domain and how we would frame it as a POMDP problem.

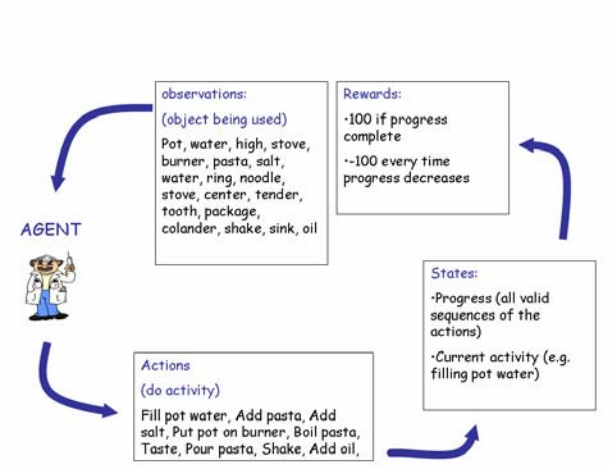

Figure 11: Example of a prompting scenario where an elder person boils pasta. The system monitors the person, diagnoses his/her state and prompts him/her appropriately to complete the task. This is a non-platform autonomic system where the state variables capture task and user states. This problem was studied by the authors in collaboration with the Intel Lablet in Seattle.

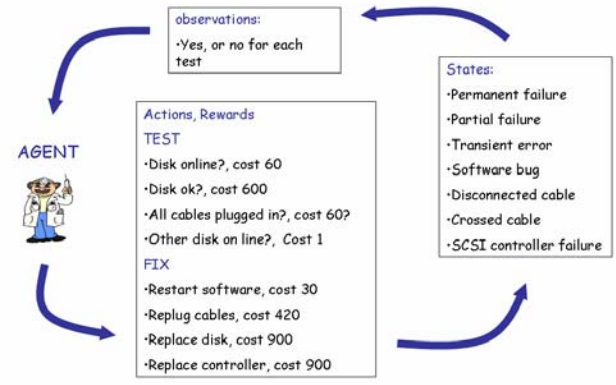

Figure 12: Example of an autonomic system for diagnosing a failed disk on a server and repairing it. It was modeled as a POMDP in [9]. This is a classic platform autonomic system where the user is not explicitly considered.

There are many other similar problems of current interest to Intel that can naturally fit the model-based decision-making framework. Among these are the following:

- *Identity capable platforms*. In this domain the idea is to develop trusted access across a variety of devices, networks, and services. This can be achieved by identity-based policies where identity could be the user itself, or some mode of operation. This is obviously a system that needs to explicitly reason about the user in order to achieve some long-term goals, such as ensuring a trusted network or certain services.
- *Server power management*. In this domain power management has to be done over an entire server farm. The system state is captured by sensors, which monitor the work load, the temperature, and the different processes that run on each server. The actions manage the system by assigning workloads dynamically to servers. The reward function defines the objective of the system by providing a positive reward when the power (and heat) distributions are as balanced as possible and jobs get executed without delays; otherwise, the reward is negative.
- *Virus detections on networks*. In this domain an agent has to predict whether or not there is a worm attack based on local network traffic patterns. The state here represents the network and traffic patterns. There are no actions here but rather the agent has to decide whether or not the network is currently under attack. The reward is positive when attacks are correctly identified and negative otherwise.
- *Memory power management*. In this domain, the agent needs to predict whether blocks of RAM can be turned off without impacting the performance that is perceived by the user. The system and user state here are similar to the APM domain. Actions are whether to turn on or off blocks of RAM. The reward function represents user's annoyance and power costs similar to the APM problem.
- *Awareness CPU architecture*. In this domain the agent needs to forecast the next upcoming CPU state based on usage patterns. This is again a very similar problem to the OS power management problem except that it needs to be done at the millisecond level.
- *Multi-device transparent power sharing*. In this domain machines need to automatically share power. Given potential usage, benefit, and activity of each device, power should be divided according to how the users' tasks are accomplished and should not take into account individual device power levels.

CONCLUSION

In this paper, we described how a machine-learning methodology could be applied to APM. Despite the simplifying assumptions we made about observability and temporal dynamics, the performance of our approach is better than naïve classification techniques and commercially available methods. Our success is partially due to the fact that we modeled the user context, which is

a fundamental concept that arises in any autonomic domain. We are currently working on more complex models and on methods for automatic discovery of context and learning of temporal dynamics.

We believe that stochastic models such as MDPs, DBNs, and POMDPs are promising techniques for mitigating the complexity and handling uncertainty for autonomic systems. The POMDP model in particular facilitates reasoning about uncertainty and information; it allows the application designer to control the level of abstraction needed and to choose the appropriate tradeoff between model accuracy and feasibility. Yet, the POMDP model can be solved, at least sub-optimally, using relatively simple methods. Thus, the main challenge in applying the POMDP model in autonomics is not computational; rather it is a modeling challenge, i.e., creating models that are simple yet effective for the application at hand. We are currently working on using stochastic models, and POMDPs in particular, for APM.

Handling uncertainty and hidden information is at the core of many autonomic systems. Whether the designer of the autonomic system chooses a direct approach or whether a probabilistic model is preferred, the explicit consideration of the user is crucial for the success of many such systems. We believe that many autonomic systems of interest would benefit from addressing uncertainty and unobservable dynamics directly.

ACKNOWLEDGMENTS

We thank Leslie Pack Kaelbling from MIT for helpful discussions and our reviewers for valuable suggestions that significantly improved this paper.

REFERENCES

[1] Benini, L., Bogliolo, A., Paleologo, G. A., and De Micheli, G., "Policy Optimization for Dynamic Power Management," *IEEE Transactions on Computer-Aided Design*, Vol. 18, Issue 6, pp. 813–833, 1999.

[2] Bertsekas, D.P., "Dynamic Programming and Optimal Control," *Athena Scientific*, 3rd edition, 2005.

[3] Billings, D., Papp, D., Schaeffer, J., and Szafron, D., "Opponent Modeling in Poker," in *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pp. 493–499, 1998.

[4] Carmel, D. and Markovitch, S., "Incorporating Opponent Models into Adversary Search," in *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pp. 120–125, 1996.

[5] Duda, R. O., Hart, P. E., and Stork, D. G., *Pattern Classification* (2nd ed.), New York: John Wiley & Sons, Inc., 2001.

[6] Duong, T., Bui, H., Phung, D., and Venkatesh S., "Activity Recognition and Abnormality Detection with the Switching Hidden Semi-Markov Model," *International Conference on Computer Vision & Pattern Recognition*, 2005.

[7] Hwang, C. H. and Wu, A., "A Predictive System Shutdown Method for Energy Saving of Event-Driven Computation," in *Proceedings of the International Conference on Computer Aided Design*, Morgan Kaufmann, San Francisco, CA, pp. 28–32, 1997.

[8] Kaelbling, L. P., Littman, M. L. and Cassandara A. R., "Planning and Acting in Partially Observable Stochastic Domains," *Artificial Intelligence*, 101, pp. 99–134, 1998.

[9] Littman, M., Nguyen, T., Hirsh, H., Fenson, E. M., and Howard, R., "Cost-Sensitive Fault Remediation for Autonomic Computing," *International Joint Conference on Artificial Intelligence*, 2003.

[10] Madani, O., Hanks, S., and Gordon, A., "On the Undecidability of Probabilistic Planning and Infinite Horizon Partially Observable Markov Decision Processes," in *Proceedings of the Seventeenth National Conference in Artificial Intelligence*, pp. 409–416 1999.

[11] Papadimitriou, C. and Tsitsiklis, J.,"The Complexity of Markov Decision Processes," *Mathematics of Operation Research*, Vol. 12, Issue 3, 1987.

[12] Ren, Z. and Krogh, B.H., "Hierarchical Adaptive Dynamic Power Management," *IEEE Transactions on Computer*, Vol. 54, Issue 4, pp. 409–420, 2005.

[13] Shani, G., Brafman, R., and Heckerman, D. "An MDP-based recommender system," in *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, pp. 453–460, 2002.

[14] Simunic, T., Benini, L., Glynn, P., and De Micheli, G., "Event-Driven Power Management," *IEEE Transactions on CAD*, Vol. 20, Issue 21, pp. 840–857, 2001.

[15] Simunic, T., "Dynamic Management of Power Consumption," in R. Graybill and R. Melhem, editors, *Power Aware Computing*, Chapter 6, Kluwer Academic, 2002.

[16] Srivastava, M., Chandrakasan, A., and Brodersen, R., "Predictive System Shutdown and Other Energy Efficient Programmable Computation," *IEEE*

Transaction on VLSI Systems, Vol. 4, Issue 1, pp. 42–55, 1996.

AUTHORS' BIOGRAPHIES

Georgios Theocharous received his Ph.D. degree in Computer Science in 2002 from Michigan State University. From 2002 to 2004 he was a post-doctoral associate at the Computer Science and Artificial Intelligence Lab at M.I.T., and in October 2004 he joined Intel as a research scientist. His research interests include computational models of learning and planning under uncertainty and their applications to the real world. Specific models include reinforcement learning, completely and partially observable Markov decision processes (POMDPs), semi-Markov decision processes, hierarchical POMDPs, and dynamic Bayesian nets. His e-mail is georgios.theocharous at intel.com.

Shie Mannor received a Ph.D. degree in Electrical Engineering from the Technion-Israel Institute of Technology in 2002. From 2002 to 2004, he was a postdoctoral associate at the Laboratory for Information and Decision Systems at M.I.T. Since 2004 he has been an Assistant Professor of Electrical and Computer Engineering at McGill University. He was a Fulbright scholar in 2002, and he is currently a Canada Research Chair in Machine Learning. His research interests include machine learning, planning and control under uncertainty, multi-agent systems, and he has a particular interest in applications of machine learning in networks and information technology. His e-mail is shie.mannor at mcgill.ca.

Nilesh N. Shah is the principal investigator of the User Activity-Based Adaptive Power Management research project. His research is positioned at the intersection of mobile platforms, power management, machine learning, and activity inference. Shah has held several positions within Intel. Most recently, he managed an Advanced Platform Development team within the Mobility Group. He played an expatriate role as manager, helping to build the team and provide the capability for Intel's Communications Group in Shanghai, China, towards the development of network processors, WLAN/WiMAX chipsets, Ethernet, and optical switches. He joined Intel in 1998 as part of the Chipset Development group after graduating from Purdue University with a Master's degree in Electrical Engineering. His e-mail is nilesh.n.shah at intel.com.

Prashant Gandhi received his M.S. degree in Electrical Engineering in 2005 from Santa Clara University. He joined Intel as a software integrator in February 2006. His interests include mobile power management and power optimized software. Specifically he is interested in investigating the tradeoffs between performance and power for multi-threaded applications optimized for multi-core processors. His e-mail is prashant.gandhi at intel.com.

Branislav Kveton is a Ph.D. student in the Intelligent Systems Program at the University of Pittsburgh. After defending his dissertation in the Fall of 2006, he will join the Corporate Technology Group (CTG) at Intel Corporation as a full-time employee. His major interests are solving large-scale stochastic decision problems and real-world anomaly detection. His long-term goal is to keep bridging the gap between theory and the complexity of real-world problems. His e-mail is bransislav.kveton at intel.com.

Sajid Mahmood Siddiqi is a Ph.D. student in Robotics in the School of Computer Science in Carnegie Mellon University, where he received his M.S. degree in Robotics in 2005. His Ph.D advisors are Geoffrey J. Gordon and Andrew W. Moore. He received a B.S. degree in Computer Science and a B.A. degree in Mathematics and Economics from the University of Southern California in 2003. His research focuses on probabilistic and statistical methods for modeling uncertainty, particularly in time series data. In the summer of 2006, Sajid was an intern at Intel Research working with the Adaptive Power Management team, where his mentor was Georgios Theocharous and his manager was Nilesh Shah. His e-mail is siddiqi at cs.cmu.edu.

Chih-Han Yu has been a Ph.D. student in Computer Science at Harvard University since 2005. From 2003 to 2005, he worked on his M.S. degree and conducted research in the Artificial Intelligence Lab of Stanford University. Prior to that, he received his B.S. degree from the National Taiwan University, Taipei, Taiwan. In the summer of 2006, he was a research intern in Intel Corporation. His research interests primarily lie in machine learning and artificial intelligence. In particular, he is interested in applying reinforcement learning and graphical model techniques to operating systems, distributed systems, and robotics. His e-mail is chyu at fas.harvard.edu.

THIS PAGE INTENTIONALLY LEFT BLANK

For further information visit:

developer.intel.com/technology/itj/index.htm